Projektne naloge

Porazdeljeni sistemi

Uroš Lotrič, Davor Sluga

Jesen 2021

Splošne informacije

V nadaljevanju dokumenta so opisane štiri različne projektne naloge. Pričakujeva, da boste projektne naloge delali v parih. Za vsako drugačno obliko dela se morate dogovoriti z nama. Vsako nalogo si lahko izberejo tri skupine. Izbor nalog bo potekal preko učilnice po pravilu kdor prej pride, prej melje.
Za pozitivno oceno iz projektne naloge:

- pripravite sekvenčno različico algoritma, ki bo služila za primerjavo;
- pripravite paralelno različico algoritma s tehnologijami, predlaganimi pri posamezni nalogi;
- naredite primerjalne teste na superračunalniku NSC ali v oblaku (prva in druga naloga);
- pripravite 10-minutno predstavitev, ki opisuje vašo rešitev, rezultate meritev, kot so čas izvajanja, pohitritve, učinkovitosti, in glavne ugotovitve.
- predstavitev in izvorno kodo oddajte preko učilnice.

Nekaj idej za boljšo oceno:

- pripravite izvedbo algoritma za tehnologijo, ki ni eksplicitno navedena pri nalogi;
- algoritem izvajajte na več enotah (grafične kartice);
- izpeljite teoretični model paralelnega algoritma;
- predlagajte in preizkusite dodatne optimizacije algoritma.

Rok za oddajo predstavitev je **TBD**. H končni oceni projektne naloge prispevajo opravljeno delo in predstavitev.

1 Veriženje blokov

Sistem razpršenega veriženje blokov omogoča varno in učinkovito upravljanje seznama lastnikov nečesa digitalnega, ne da bi za to potrebovali osrednjo monopolno agencijo, ki bi ji morali vsi zaupati [1]. Ideja, kako strukturno preprečiti goljufanje in nekontrolirano kopiranje digitalne lastnine (denarja), temelji na dokazu opravljenega dela, ki ga sistem zahteva na nekaj minut. To opravljeno delo mora biti tako veliko, da praktično ni mogoče goljufati, saj bi moral potencialni goljuf opraviti bistveno več dela, kot vsi drugi uporabniki sistema skupaj. Protokol usklajevanja, ki temelji na prikazu opravljenega dela, izhaja iz izvajanja zapletenih računskih nalog, ki zahtevajo veliko računske moči, njihovo rešitev pa je enostavno preveriti in tako potrditi, da je bilo delo dejansko opravljeno in ni prišlo do prevare.

Ključen element usklajevanja je preverjanje istovetnosti podatkov. To se izvaja preko digitalnega prstnega odtisa ali zgoščene vrednosti (angl. hash), ki poljubno dolgemu sporočilu priredi zaporedje znakov nespremenljive dolžine. Z vsako, še tako majhno spremembo podatkov v sporočilu, se spremeni tudi njihova zgoščena vrednost.

Pri veriženju blokov se sporočila o transakcijah med uporabniki oblikujejo v bloke. Vsi, ki sodelujejo pri veriženju blokov, sledijo novim sporočilom o izvedenih transakcijah in na podlagi zgoščene vrednosti zadnjega veljavnega bloka, predloga novega bloka in nekaj naključnih dodanih znakov ali skovanke, izračunavajo novo zgoščeno vrednost. Z dodajanjem poljubno dolge skovanke lahko zgoščeno vrednost novega bloka spreminjamo tako, da ustreza predpisanim zahtevam. Ko najdemo ustrezno zgoščeno vrednost, predlog bloka, v katerem so zabeležena nova sporočila o transakcijah, razpošljemo drugim uporabnikom v odobritev. Uporabniki sistema glasujejo za veljavnost novega bloka tako, da ga po preverjanju dodajo v verigo prejšnjih, že potrjenih. Za dopolnitev verige dobimo nagrado, nato se zgodba začne od začetka.

1.1 Zgoščena vrednost in dokaz opravljenega dela

Za sporočila, sestavljena iz niza znakov, bomo zgoščene vrednosti računali z algoritmom MD5 [2]. Algoritem MD5 [3] za niz znakov vrne zgoščeno vrednost dolžine 16 znakov. Zgoščeno vrednost nato predstavimo z 32 znaki šestnajstiške kode in preštejemo s koliko ničlami se konča. Na primer, algoritem MD5 za vhodne znake $01020304C2AAD20D_{\text{hex}}$ izračuna zgoščeno vrednost $B18F3F414358E488AEFABE8010000000_{\text{hex}}$ s 7 ničlami na koncu.

Osredotočili se bomo na dokazovanje opravljenega dela. Na vhodu pričakujemo množico sporočil različnih dolžin, na primer $01020304C2AAD2_{\text{hex}}$, $01020304C2_{\text{hex}}$, 01020304_{hex}. Vsako sporočilo posebej dopolnimo s skovanko, da dobimo zgoščeno vrednost z zahtevanim številom ničel na koncu, na primer 7. Za navedena sporočila bomo z intenzivnim preiskovanjem poiskali skovanke $0D_{\text{hex}}$, $AAD20D_{\text{hex}}$ in $C2AAD20D_{\text{hex}}$.

Kodo v jeziku C za iskanje skovanke za eno sporočilo najdete v [4]. Koda uporablja funkcije MD5 za postopno računanje zgoščene vrednosti iz knjižnice OpenSSL.

1.2 Naloga

Ker želimo dobiti veliko nagrad, bomo iskanje zgoščenih vrednosti računali vzporedno. Sporočila obdelujemo zaporedno, enega za drugim. Najti želimo najkrajšo skovanko, s katero dobimo zahtevano zgoščeno vrednost, zato skovanke podaljšujemo postopno, znak po znak. Najprej preverimo osnovno sporočilo. Če ne vrne ustrezne zgoščene vrednosti, preverimo prostor eno-znakovnih skovank. Če sporočilo z nobeno skovanko ne da zahtevane zgoščene vrednosti, začnemo preiskovati prostor dvo-znakovnih skovank. Skovanke podaljšujemo, dokler ne najdemo ustrezne skovanke ali presežemo največjo dovoljeno dolžino.

Za računanje skrbi koordinator, ki sprejema sporočila in razdeljuje delo delavcem. Predpostavite lahko, da je število delavcev omejeno na 256. Tako koordinator delo razdeli samo glede na prvi znak skovanke. Pri štirih delavcih bi prvemu dodelili preverjanje skovank s prvim znakom $00..3F_{\text{hex}}$, drugemu skovanke, ki se začnejo z znaki $40..7F_{\text{hex}}$, tretjemu skovanke s prvimi znaki $80..BF_{\text{hex}}$ in zadnjemu skovanke s prvimi znaki $C0..FF_{\text{hex}}$.

Takoj, ko delavec najde rešitev, mora o tem obvestiti koordinatorja, ki pošlje zahtevo za prekinitev iskanja ostalim delavcem. Ko vsi delavci prekinejo z iskanjem, jim koordinator dodeli naslednjo nalogo.

V osnovni različici je število delavcev znano v naprej. Predpostavite, da je sistem stabilen – koordinator in delavci vestno opravljajo svojo nalogo, z omrežjem ni težav.

Rešitev zasnujte večnitno z uporabo knjižnice pThread in porazdeljeno s knjižnico MPI. Primerjajte čase sekvenčnega preiskovanja z večnitno in porazdeljeno rešitvijo. Pri programiranju si lahko pomagate s koncepti iz navodil [5; 6].

Nekaj idej:

- Kako se čas računanja, pohitritev in učinkovitost spreminjajo s številom zahtevanih ničel na koncu zgoščene vrednosti?
- Izmerite čas, ki poteče od takrat, ko eden od delavcev najde rešitev, do takrat, ko začnejo obdelovati naslednje sporočilo.
- Sistem dopolnite tako, da delo dobijo tudi delavci, ki v sistem pristopijo med iskanjem rešitve za neko sporočilo. Uporabite idejo dinamičnega razvrščanja dela, ki smo jo srečali pri OpenMP.
- Pripravite rešitev za grafične procesne enote, pri tem se ne omejujte na 256 delavcev.
- Problem rešite z nalogami v OpenMP (deli in vladaj). Rešitev primerjajte z vašo rešitvijo s knjižnico pThread.
- Porazdeljeni sistem (MPI) naredite robusten, da bo našel rešitev tudi v primeru, ko eden od delavcev odpade.
- Dodajte odjemalce, ki pošiljajo delo koordinatorju. Koordinator naj rezultat posreduje pravemu odjemalcu.

2 Stiskanje podatkov

Stiskanje podatkov je pomembno za učinkovito prenašanje sporočil. Stiskanje podatkov, ki ga uporabljajo telefaksi pri pošiljanju slik in dokumentov preko telefonskega omrežja [11], se začne z verižnim kodiranjem (angl. run-length encoding), ki mu sledi Huffmanovo kodiranje.

2.1 Postopek stiskanja

Črno-bela stran A4 je sestavljena iz 1145 vrstic s po 1728 slikovnimi točkami. Kodiranje izvajamo nad vsako vrstico posebej. Vrstica je predstavljena z zaporedjem verižnih kod, ki se izmenjujejo za bele in črne skupine slikovnih točk. Opis vrstice se vedno začne z verižno kodno zamenjavo za bele slikovne točke. Če je prva slikovna točka v vrstici črna, najprej pošljemo verižno kodno zamenjavo dolžine nič za belo slikovno točke. Standard predvideva ločene kodne zamenjave za dolžine od 0 do 63, vse nadaljnje kodne zamenjave pa so sestavljene iz dveh delov: večkratnika števila 64 in razlike do končne dolžine, na primer $1266 = 1216 + 50$. V takih primerih si sledita dve kodni zamenjavi za isto barvo. Za lažjo sinhronizacijo ob napakah pri prenosu se vrstica vedno zaključi z unikatno kodo EOL.

Huffmanovo kodiranje se izvaja nad verižnimi kodnimi zamenjavami, ločeno za bele in črne slikovne točke. Huffmanove kodne zamenjave so bile določene na podlagi statistične analize večjega števila dokumentov. S Huffmanovimi kodnimi zamenjavami v vsaki vrstici nadomestimo verižne kodne zamenjave belih in črnih slikovnih točk. Kodna zamenjava za EOL je enaka 000000000001 in ni kodirana s Huffmanovim algoritmom. Postopek kodiranja je nakazan na povezavi [12].

Standard lahko izboljšamo tako, da za vsako stran A4 zgradimo optimalni Huffmanov kod za bele in črne verižne kodne zamenjave. Huffmanov kod vrne optimalne dolžine kodnih zamenjav za podane osnovne znake z znanimi verjetnostmi. V našem primeru osnovne znake predstavljajo verižne kodne zamenjave, verjetnosti pa dobimo iz števila ponovitev verižnih kodnih zamenjav. Algoritem je skiciran na sliki 1, lahko pa si pomagate tudi s kodo [13; 14].

2.2 Naloga

Na voljo imamo veliko količino črno-belih dokumentov (strani A4), ki jih želimo zakodirati po zgoraj opisanem postopku. Postopek želimo čim bolj pohitriti. Stiskanje strani A4 lahko razdelimo na štiri korake: (i) branje bitne slike strani A4, (ii) verižno kodiranje, (iii) Huffmanovo kanonično kodiranje in (iv) zapisovanje kode strani v datoteko. Posamezne korake težko paraleliziramo, za pohitritev je v tem primeru najbolj enostavno uporabiti štiri-stopenjski cevovod. Počasnejše stopnje lahko vzporedno izvajamo na več procesorskih jedrih.

Rešitev zasnujte okrog knjižnice pThread, komunikacijo med stopnjami pripravite po konceptu proizvajalec-porabnik. če vam je ljubše, lahko poskusite z jezikom go ali kotlin.

Prvi del:
1. Ustvarimo seznam osnovnih znakov L(1..n) s pripadajočimi verjetnostmi.
2. Ponavljajmo, dokler ne pridemo do sestavljenega znaka z verjetnostjo 1:
   - v seznamu L poiščimo znaka z najmanjšima verjetnostima;
   - ustvarimo nov sestavljen znak, z verjetnostjo, enako vsoti verjetnosti znakov;
   - najdena znaka označimo kot uporabljena;
   - nov sestavljen znak dodamo v seznam L;
   - zabeležimo si, iz katerih znakov je sestavljen.
3. Za vse sestavljene znake v L (od zadnjega proti prvemu):
   - znak razstavimo – poišečemo oba znaka, ki ga sestavljata;
   - obema sestavnima deloma povečamo dolžino kodne zamenjave za 1.

Drugi del:
1. Uredimo znake po dolžinah kodnih zamenjav (KZ) in nato po njihovi vrednosti (naraščajoče).
2. KZ $\longleftarrow$ zeros(1, dolžina KZ); prvi znak je sestavljen iz samih ničel
3. Dokler so znaki na vhodu:
   - izstavi znak in KZ;
   - KZ $\longleftarrow (KZ + 1) <<$ ((dolžina naslednje KZ) - (dolžina trenutne KZ));

Slika 1: Huffmanov algoritem.

V osnovni različici ni potrebno, da je vrstni red dokumentov na izhodu enak kot na vhodu. Nekaj idej:

- Izmerite čase procesiranja na posameznih stopnjah cevovoda, izračunajte pohitritve, učinkovitost.
- Koncept proizvajalec-porabnik razširite tako, da bo proizvajalec naloge lahko oddajal v medpomnilnik.
- Preverite, koliko pomaga uporaba več vzporednih procesorjev v izbranih stopnjah cevovoda.
- Za koliko se zmanjšajo pohitritve, če zahtevate, da je vrstni red dokumentov na izhodu enak tistemu na vhodu?
- Na drugem vozlišču pripravite cevovodno rešitev za dekodirnik, pošiljanje podatkov izvedite preko MPI ali telegramov TCPIP.
- Program izboljšajte tako, da število vzporednih procesorjev na vsaki stopnji določate dinamično glede na potrebe.

3 Stiskanje slik s pomočjo razvrščanja z voditelji

Vseprisotnost multimedijskih vsebin na spletu močno obremenjuje internetno infrastrukturo [28]. Da bi zmanjšali količino prenesenih podatkov se poslužujemo raznovrstnih postopkov za izgubno [23] in brezizgubno [24] stiskanje takih vsebin. Tukaj si bomo ogledali postopek stiskanja s pomočjo razvrščanja z voditelji [25; 26], ki spada med metode nenadzorovanega učenja.

Razvrščanje z voditelji

Razvrščanje z voditelji je metoda, namenjena razvrščanju vzorcev v skupine. Zaradi svoje enostavnosti in robustnosti je zelo razširjena. Za osnovni algoritem je bilo razvitih tudi precej izboljšav in prilagoditev namenjenih reševanju različnih problemov [27].

Recimo, da imamo podano množico n vzorcev $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n)$, kjer je vsak vzorec D-dimenzionalni vektor. Te vzorce želimo razvrstiti v k skupin ali gruč $\mathbf{C} = \{C_1, C_2, C_3, \ldots, C_k\}$, kjer je $k \leq n$. Poleg same razvrstitve želimo izračunati še centroide $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$, ki predstavljajo povprečen vzorec znotraj posamezne gruče. Razvrščanje v gruče želimo opraviti tako, da minimiziramo raznolikost oziroma varianco znotraj gruče:

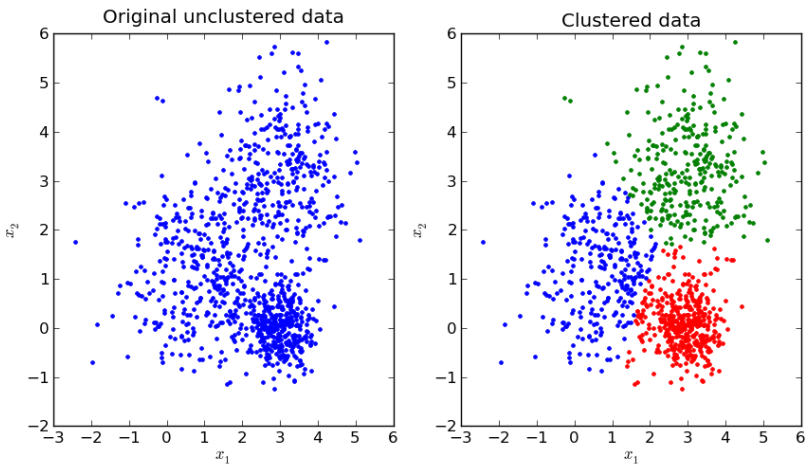
$$\arg\min_{\mathbf{C}} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 \quad . \tag{1}$$

Najbolj običajen algoritem za reševanje tega problema je metoda iterativnega izboljševanja:

Inicializiraj začetne vrednosti centroidov $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$ tako, da jim prirediš naključen vzorec $\mathbf{x}_i$
while $I <$ Iteracij **do**
  for vsak vzorec $\mathbf{x}_i$ **do**
    poišči $\boldsymbol{\mu}_j$ z najmanjšo evklidsko razdaljo do $\mathbf{x_i}$
    nastavi indeks centroida $c_i = j$
  end for
  for vsak centroid $\boldsymbol{\mu}_j$ **do**
    izračunaj povprečje $\mathbf{m}$ vseh $\mathbf{x}_i$, kjer $c_i == j$
    nastavi $\boldsymbol{\mu}_j = \mathbf{m}$
  end for
end while
Polje c vsebuje indekse centroidov za vse vzorce
$\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$ so končne vrednosti centroidov

Slika 2: Algoritem razvrščanja z voditelji.

Pri razvrščanju se lahko zgodi, da katera izmed gruč ostane prazna. Temu se običajno želimo izogniti. To lahko storimo tako, da prazni gruči pripišemo tisti vzorec, ki je najdlje od svoje trenutne gruče.



Slika 3: Primer razvrščanja 2D vzorcev v tri gruče.

Stiskanje slike

Za zapis pikslov neke slike lahko uporabimo predstavitev RGBA (angl. red, green, blue, alpha) pri čemer uporabimo 8 bitov za zapis intenzitete posameznega barvnega kanala (razpon vrednosti 0–255), skupaj torej 32 bitov na piksel. Število bitov na piksel lahko zmanjšamo, če zmanjšamo število različnih barv v sliki. Naiven pristop je, da enostavno zmanjšamo število razpoložljivih nivojev intenzitete za vsak barvni kanal (na primer iz 256 na 16, kar nam da 16 bitov na piksel) in ustrezno preračunamo nove vrednosti intenzitet. Ta pristop lahko precej pokvari kvaliteto slike. Da bi bolje ohranili informacijo v sliki, je potrebno upoštevati porazdelitev barv v njej.

Ena od možnosti je, da uporabimo razvrščanje z voditelji in podobne barve združimo v gruče, katerih centroide nato uporabimo kot nadomestno barvo za vse piksle znotraj gruče. V tem primeru so naši vzorci piksli - vektorji v štiri-dimenzionalnem prostoru RGBA. Ciljno število različnih barv, ki jih želimo uporabiti v sliki, pa je število gruč v katere razvrščamo piksle. S takim pristopom učinkovito zmanjšamo število različnih barv v sliki in posledično njeno velikost ter bolje ohranimo kvaliteto slike.

(a) Vhodna slika: 32 bitov na piksel.

(b) Izhodna slika: 64 različnih barv (6 bitov na piksel).

Slika 4: Stiskanje slike s pomočjo razvrščanja z voditelji.

Naloga

S pomočjo OpenCL in pthreads/OpenMP napišite vzporedno različico algoritma za stiskanje slik s pomočjo razvrščanja z voditelji. Program naj omogoča stiskanje slik poljubnih velikosti za poljubno število barv. Program naj slike bere in zapisuje v formatu PNG. Za branje in zapisovanje slik lahko uporabite ustrezno knjižnico, na primer STB [29] ali FreeImage [21]. Program naj omogoča izbiro števila niti, ki bodo uporabljene pri procesiranju (OpenMP/pthreads).

Nekaj idej:

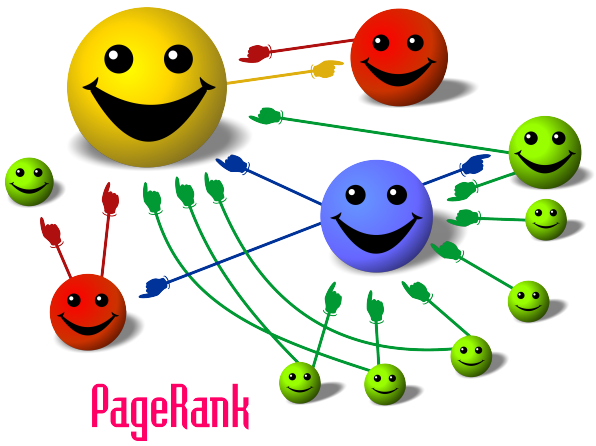
- Algoritem ustrezno preoblikujte, da zmanjšate število dostopov do pomnilnika.
- Mogočih je več pristopov k paralelizaciji algoritma. Med niti lahko razdelite vzorce ali pa gruče. V večini primerov je prvi pristop boljši, saj je na voljo veliko več vzorcev kot gruč.
- Pazite na razporeditev dela, nekatere gruče lahko vsebujejo veliko več vzorcev kot druge.
- Optimizirajte število niti na OpenCL (skupno število niti, število niti na blok).
- Uporabite več grafičnih kartic hkrati.

4 Pagerank

Algoritem Pagerank [22] sta leta 1998 predstavila Larry Page in Sergey Brin, ustanovitelja podjetja Google. Pagerank je uporabljal in ga v določeni meri še vedno uporablja spletni iskalnik Google pri rangiranju spletnih strani med prikazanimi zadetki za nek iskalni termin. Algoritem pri razvrščanju strani upošteva dva glavna faktorja, ki vplivata na pomembnost neke spletne strani:

- Število strani, ki vsebujejo povezavo do dane spletne strani.
- Pomembnost strani, ki vsebujejo povezave do dane spletne strani.

Drugače povedano, do neke strani lahko vodi razmeroma malo povezav, vendar se le-te nahajajo na pomembnih, oziroma zelo znanih straneh (e.g. nytimes.com), kar pomeni, da bo dana stran visoko rangirana. Lahko pa do neke strani vodi veliko število povezav iz manj pomembnih virov (razni blogi, forumi, ...), kar ji bo prav tako prineslo visok rang.



Matematično lahko zgornjo idejo zapišemo z naslednjo formulo:

$$R(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{R(p_j)}{L(p_j)}.$$

V zgornji enačbi je $R(p_i)$ rang spletne strani p_i, ki ga iščemo; $M(p_i)$ je množica strani, ki vsebujejo povezavo do strani p_i; $L(p_j)$ je število izhodnih povezav, ki jih vsebuje stran p_j in N je število vseh strani. Enačba vsebuje še teleportacijski faktor d, ki ga običajno nastavimo na 0.85. Z njim modeliramo dejstvo, da lahko posamezno stran obiščemo po naključju, brez da bi do nje vodila povezava iz neke druge strani.

PageRank lahko izračunamo iterativno. Algoritem doseže konvergenco precej hitro. V izvornem članku [22] avtorja omenjata, da je za podatkovno bazo, ki vsebuje 322 milijonov povezav potrebnih 52 iteracij.

Postopek:
1. Nastavimo t=0
2. Nastavimo začetno vrednost $R(p_i;t) = 1/N$ za vse spletne strani.
3. Ponavljamo, dokler ne dosežemo ustavitvenega pogoja $||R(\mathbf{p};t+1) - R(\mathbf{p};t)|| < \epsilon$:

$$R(p_i;t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{R(p_j;t)}{L(p_j)}$$

Slika 5: Algoritem za izračun PageRank.

Iterativno enačbo v koraku 3, lahko zapišemo tudi v matrični notaciji:

$$\mathbf{R}(t+1) = d\mathbf{M}\mathbf{R}(t) + \frac{1-d}{N}\mathbf{1},$$

kjer je $\mathbf{R}(t)$ stolpični vektor rangov v koraku t, $\mathbf{1}$ stolpični vektor samih enic, $\mathbf{M}$ pa matrika definirana kot

$$\mathbf{M}_{ij} = \begin{cases} L(p_j), & p_j \to p_i \\ 0, & \text{sicer.} \end{cases} \tag{2}$$

4.1 Naloga

Kljub temu, da je sam algoritem za rangiranje spletnih strani precej enostaven, hitro naletimo na težavo, da je število spletnih strani ogromno (trenutno že preko 50 milijard), kar seveda pomeni, da s serijskim algoritmom ne bomo prišli daleč. Ideja je torej, da pripravimo vzporedno različico algoritma, ki bo prišla do rešitve hitreje. Napišite vzporedno različico algoritma s pomočjo tehnologije OpenCL in večnitenja (pThreads ali OpenMP) in ju primerjajte s sekvenčnim algoritmom. Za testni nabor podatkov lahko uporabite bazo, ki jo je objavil Google (https://snap.stanford.edu/data/web-Google.html).

Nekaj Idej:

- Primerjajte učinkovitost in pohitritev za različno število vozlišč.
- Razmislite in pripravite učinkovit način hranjenja podatkovnih struktur, ki opisujejo povezave med stranmi.
- Pri procesiranju uporabite več grafičnih kartic.
- Primerjajte vzporedno rešitev na GPU z vzporedno rešitvijo na CPU (časi, pohitritve)

Literatura

[1] Kvarkadabra, Kaj je blockchain?, https://kvarkadabra.net/2016/10/kaj-je-blockchain/, 2021.

[2] MD5, https://en.wikipedia.org/wiki/MD5, 2021.

[3] Openwall, A portable, fast, and free implementation of the MD5 Message-Digest Algorithm (RFC 1321), https://openwall.info/wiki/people/solar/software/public-domain-source-code/md5, 2021.

[4] Lotrič, U., Iskanje skovanke za dano sporočilo, https://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=47001, UL FRI, 2021.

[5] Beschastnikh, I., 416 Distributed Systems: Assignment 2 [Proof-of-work], https://www.cs.ubc.ca/~bestchai/teaching/cs416_2020w2/assign2/index.html, University of British Columbia, 2021

[6] Beschastnikh, I., 416 Distributed Systems: Assignment 3 [Distributed proof-of-work], https://www.cs.ubc.ca/~bestchai/teaching/cs416_2020w2/assign3/index.html, University of British Columbia, 2021.

[7] Wikipedia, https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life, 2020

[8] S. Wolfram, A new Kind of Science, Wolfram Media, https://www.wolframscience.com/nks/, 2002

[9] A. Alexandridis et. al., A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through Spetses Island in 1990, Applied Mathematics and Computation, 204, 191-201, 2008

[10] XC-Li, Python koda po članku A. Alexandridis et. al., https://github.com/XC-Li/Parallel_CellularAutomaton_Wildfire, 2020

[11] CCITT, Protocols for Telematic Services-Standardization of Group 3 facsimile apparatus for document transmission, ITU-T T.4, https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.4-200307-I!!PDF-E&type=items, 1999

[12] Modified Huffman coding, Wayback machine Internet Archive, https://web.archive.org/web/20020628195336/http://www.netnam.vn/unescocourse/computervision/104.htm, 2020

[13] M. Dipperstein, Huffman Code Discussion and Implementation, https://michaeldipperstein.github.io/huffman.html, 2020

[14] W. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd Edition, Cambridge University Press, USA, 2007

[15] Goyal, A., Bijalwan, A., Chowdhury, M. K. (2012). A comprehensive review of image smoothing techniques. International Journal of Advanced Research in Computer Engineering and Technology, 1(4), 315-319. http://ijarcet.org/wp-content/uploads/IJARCET-VOL-1-ISSUE-4-315-319.pdf

[16] Brown University, https://cs.brown.edu/courses/csci1290/labs/lab_bilateral/index.html, 2020

[17] Intel OneAPI, https://software.intel.com/content/www/us/en/develop/tools/oneapi.html, 2020

[18] Intel Dev Cloud, https://software.intel.com/content/www/us/en/develop/tools/oneapi/all-toolkits.html, 2020

[19] James Gregson's Website, http://jamesgregson.ca/bilateral-filtering-in-python.html, 2020

[20] Parallel Bilateral Filtering using OpenCL, http://xidexia.github.io/Bilateral-Filtering/, 2020

[21] FreeImage, https://freeimage.sourceforge.io/, 2021

[22] Page, Lawrence and Brin, Sergey and Motwani, Rajeev and Winograd, Terry (1999). The PageRank citation ranking: Bringing order to the web.

[23] Wikipedia, https://en.wikipedia.org/wiki/Lossy_compression, 2021.

[24] Wikipedia, https://en.wikipedia.org/wiki/Lossless_compression, 2021.

[25] J. MacQueen, Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297), 1967

[26] S. Lloyd, Least squares quantization in PCM. IEEE transactions on information theory Vol. 28, No. 2, pp. 129-137, 1982.

[27] Wikipedia, https://en.wikipedia.org/wiki/K-means_clustering, 2021.

[28] Arnes, https://www.arnes.si/infrastruktura/six-sticisce-omrezij/statistika-prometa/, 2021.

[29] STB library, https://github.com/nothings/stb, 2021.